

# Security Validation Attack Failure on Patched Server Documentation

By Liben Tadele

This section documents how the previously successful NoSQL injection attacks **no longer work** after applying security fixes.

The goal is to **prove** that:

- Vulnerabilities were fixed
- Attacks now fail
- Unauthorized access is blocked

---

## Purpose of This Documentation

This documentation shows that:

- NoSQL injection payloads are rejected
- Authentication bypass is no longer possible
- Sensitive data cannot be extracted
- Role-based access control (RBAC) is enforced correctly

This confirms the application is no longer vulnerable to the demonstrated attacks.

---

## Test Environment

- Server: Fixed version of the NoSQL Injection Vulnerable Server
- Database: MongoDB
- Testing Tool: Postman
- Authentication: JWT
- User Role Tested: Unauthenticated user

---

## Injection Vector 1: Authentication Bypass Attempt

### Endpoint Tested

POST /auth/login

### Payload Used (Previously Successful)

```
{  
  "username": "admin",  
  "password": "password123"  
}
```

```
"password": { "$ne": null }  
}
```

The screenshot shows a Postman interface with the following details:

- Method:** POST
- URL:** http://127.0.0.1:4000/auth/login
- Body:** Raw JSON (selected)
- JSON Payload:**

```
1  {  
2    "username": "admin",  
3    "password": { "$ne": null }  
4  }
```
- Response Status:** 400 Bad Request
- Response Body:**

```
1  {  
2    "message": "Invalid input types"  
3  }
```

## Injection Vector 2: Regex Injection in Login

### Endpoint Tested

POST /auth/login

### Payload Used

```
{  
  "username": { "$regex": ".*" },  
  "password": { "$ne": null }  
}
```

The screenshot shows the Postman interface with a POST request to `http://127.0.0.1:4000/auth/login`. The Body tab is selected, containing the following JSON payload:

```
1 {
2   "username": { "$regex": ".*" },
3   "password": { "$ne": null }
4 }
```

The response tab shows a 400 Bad Request status with the message: "Invalid input types".

## Injection Vector 3: User Enumeration via Search

## Endpoint Tested

**GET /search?q={"username":\$regex:".\*"}**

The screenshot shows the Postman application interface. At the top, the URL `http://127.0.0.1:4000/search` is entered in the address bar, along with a `GET` method indicator. To the right are buttons for `Save`, `Share`, and a `Send` button. Below the address bar, the `Params` tab is selected, showing two parameters: `q` with value `{"username": "$regex": ".+"}`. The `Headers` tab shows nine headers. In the main content area, under the `Body` tab, a JSON response is displayed with the message `"message": "Complex queries not allowed"`. The status bar at the bottom indicates a `400 Bad Request` status with a duration of `15 ms` and a size of `317 B`.

## Injection Vector 4: Sensitive Data Extraction

## Endpoint Tested

## POST /users/filter

## Payload Used

```
{ "ssn": { "$exists": true } }
```

The screenshot shows a Postman interface with the following details:

- URL: `http://127.0.0.1:4000/users/filter`
- Method: `POST`
- Body type: `raw`, `JSON`
- Body content:

```
1 { "ssn": { "$exists": true } }
```
- Response status: `400 Bad Request`
- Response message:

```
1 {  
2   |   "message": "Operators not allowed"  
3 }
```

## Injection Vector 5: Medical Records Extraction

### Endpoint Tested

`POST /records/search`

## Payload Used

```
{ "patientId": { "$regex": ".*" } }
```

The screenshot shows a Postman interface with the following details:

- Request URL:** `http://127.0.0.1:4000/records/search`
- Method:** POST
- Body Content:**

```
1 { "$patientId": { "$regex": ".+" } }
```
- Response Status:** 400 Bad Request
- Response Body:**

```
1 {  
2   "message": "Operators not allowed"  
3 }
```

## Injection Vector 6: Admin Endpoint Access Without Proper Authorization

### Endpoint Tested

POST /admin/export

### Test Condition

- No JWT token
- Invalid or non-admin token

http://127.0.0.1:4000/admin/export

POST http://127.0.0.1:4000/admin/export

Headers (10) Body Cookies

Headers (10)

Key	Value	Description
Authorization	Bearer	
Key	Value	Description

Body Cookies Headers (8) Test Results

401 Unauthorized 117 ms 304 B

{ } JSON Preview Debug with AI

```
1 {  
2 | "message": "Invalid token"  
3 }
```

## Injection Vector 7: Extract Admin Accounts

### Endpoint

GET /search

### Payload

?q={"role":"admin"}

http://127.0.0.1:4000/search

GET http://127.0.0.1:4000/search?q={"role":"admin"}

Params (10) Headers (8) Cookies

Params

Key	Value	Description
q	{"role":"admin"}	
Key	Value	Description

Body Cookies Headers (8) Test Results

400 Bad Request 7 ms 317 B

{ } JSON Preview Debug with AI

```
1 {  
2 | "message": "Complex queries not allowed"  
3 }
```

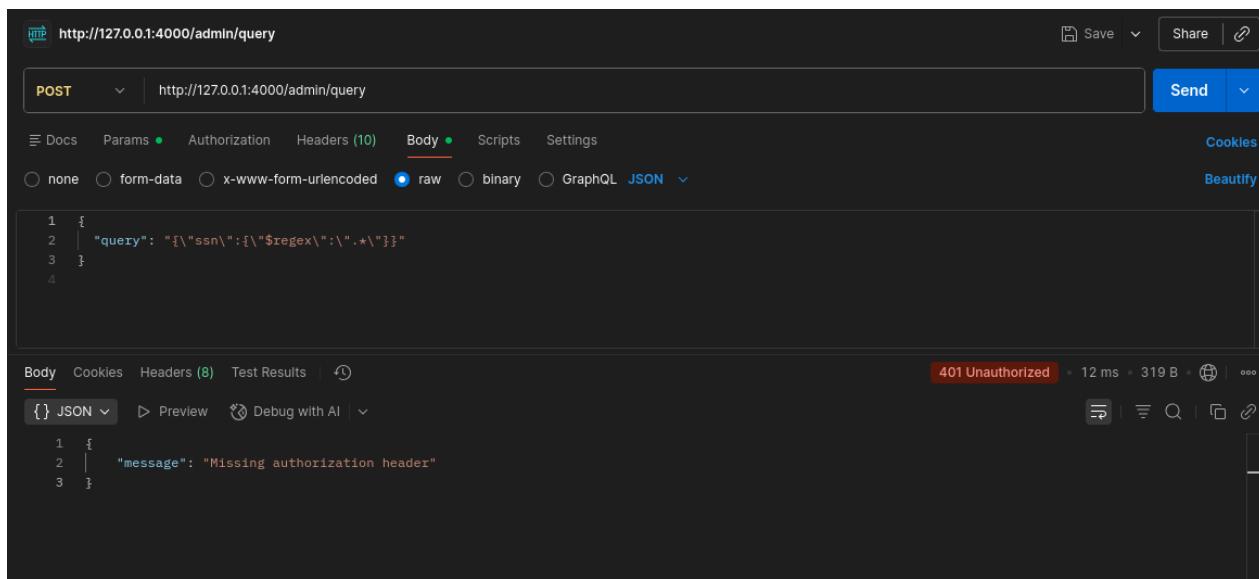
# Injection Vector 8: Admin Query Injection

## Endpoint

POST /admin/query

## Payload

```
{ "query": "{\\\"ssn\\\":{\\\"$regex\\\":\\\".*\\\"}}" }
```



# Automated Exploit Script Validation

An automated Python exploit script that previously succeeded in exploiting the NoSQL injection vulnerabilities was re-executed against the fixed server.

## Exploit Script Used

- Script: `nosql_exploit.py`
- Purpose: Automate authentication bypass and full data extraction

```
$ python3 nosql_exploit.py
=====
NoSQL Injection Exploit - Full Attack Chain
=====
target: http://127.0.0.1:4000
STAGE 1: INJECTION DISCOVERY

VULN TYPE: Authentication Bypass - Operator Injection]
INPUT: POST /auth/login
PAYLOAD: {"username": "admin", "password": {"$ne": null}}
OUTPUT: Status 400
RESULT: Failed - {"message":"Invalid input types"}

VULN TYPE: Authentication Bypass - Regex Injection]
INPUT: POST /auth/login
PAYLOAD: {"username": {"$regex": ".*"}, "password": {"$ne": null}}
OUTPUT: Status 400
RESULT: Failed

VULN TYPE: Data Extraction - Search Injection]
INPUT: GET /search
PAYLOAD: q={"$regex": ".*"}
OUTPUT: Status 400
RESULT: Failed

VULN TYPE: Privileged Data Extraction - Role-based Query]
INPUT: GET /search
PAYLOAD: q={"role": "admin"}
OUTPUT: Status 400
RESULT: Failed
STAGE 2: AUTHENTICATION BYPASS

VULN TYPE: Authentication Bypass - Password Bypass]
INPUT: POST /auth/login
PAYLOAD: {"username": "admin", "password": {"$ne": null}}
OUTPUT: Status 400
RESULT: Authentication failed
STAGE 3: DATA EXTRACTION

VULN TYPE: User Enumeration - Full Extraction]
```

## Security Validation Outcome

All tested NoSQL injection vectors that previously resulted in authentication bypass, data exposure, or privilege escalation were successfully mitigated. The application now correctly treats all user input as untrusted data and prevents it from being interpreted as database logic.

Validation testing confirms that malicious MongoDB operators are blocked, unauthorized requests are denied, and sensitive resources are accessible only to properly authenticated and authorized users. No evidence of data leakage or privilege abuse was observed during post-fix testing.

This outcome demonstrates that the applied remediation measures are effective and that the system meets basic security expectations for input validation, authentication, and access control.